SISTEMA EMBEBIDO PARA LA CLASIFICACIÓN DE IMÁGENES EN TIEMPO REAL EN APLICACIONES AGRÍCOLAS Y/O INDUSTRIALES

Luisa María Millán Duque

Proyecto de grado presentado como requisito parcial
para aspirar al título de Ingeniera Electricista

Director
Ing Germán A. Holguín L, M.Sc

Grupo de Investigación en Gestión de
Sistemas Eléctricos, Electrónicos y Automáticos.

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
PROGRAMA DE INGENIERÍA ELÉCTRICA
PEREIRA
2021

Dedicado a mis padres, mi hermana y Manolo por su amor y compañía incondicional.

Agradezco a Dios, a mis padres y hermana por el sacrificio y apoyo incondicional. Al profesor Germán A Holguín L y al grupo de investigación en Gestión de Sistemas, en especial a los profesores Byron Hernandez y Óscar Martinez; muchas gracias por la oportunidad, la paciencia, el conocimiento y el apoyo. Agradezco también al ingeniero Germán Grandas por brindarme de su valioso tiempo y ayuda en el desarrollo de este proyecto, por último a mis compañeros y profesores, gracias por hacer parte de mi proceso educativo.

CONTENIDO

pág.

LISTA DE TABLAS

LISTA DE FIGURAS

1. INTRODUCCIÓN

1.1. DEFINICIÓN DEL PROBLEMA

En los últimos años, el uso de aprendizaje automático en sistemas embebidos ha ganado atención debido a las amplias aplicaciones dentro de los campos de medicina, agricultura, industrias o universidades. El empleo de unidades de procesamiento gráfico más conocidas como GPU dentro de dichos sistemas, aceleran procesos computacionales complejos que presentan los algoritmos utilizados para reconocer un objeto o clasificar una imagen. Sin embargo, la mayoría de implementaciones han requerido de sistemas de cómputo de alta gama, lo cual demanda costos operativos y consumo de energía más altos , por lo tanto , resulta ser una limitación en términos de asequibilidad y operación en ámbitos educativos o empresariales [1].

Actualmente, las redes neuronales han adquirido mayor fuerza para realizar tareas de clasificación de imágenes. Estas destacan en el área de agricultura de precisión, alegando que pueden obtener mayor precisión en las bases de datos extensas y extraer y aprender mejor las características en los resultados, incluso pueden llegar a ser más eficientes que los algoritmo clásicos de Machine Learning, pero requieren de mucho almacenamiento en memoria y potencia computacional debido a la complejidad de los cálculos llevados a cabo por este tipo de algoritmos [2].

Muchas aplicaciones agrícolas e industriales pueden asignarse a un problema de clasificación en la visión por computadora, una de esas por ejemplo, incluso cerca de nuestra región (que es la zona cafetera) es la detección de roya en las hojas del café. Actualmente, se desarrollan modelos y sistemas de detección temprana de enfermedades en cultivos, que puedan ser de ayuda a los agricultores para evitar pérdidas económicas, pero estos siempre requieren de bases de datos muy extensas lo cual conlleva a uso

de equipos sofisticados, dificultando así su funcionamiento en ubicaciones remotas y generando costos computacionales altos.

Por tal motivo, existe la necesidad de explorar el uso de algoritmos de machine learning y deep learning sobre sistemas embebidos, siendo esto el tema central de este trabajo, que plantea el uso de algoritmos de aprendizaje supervisado para llevar a cabo una tarea de clasificación de imágenes, que ayude a los productores de café a detectar la roya en el sitio y casi en tiempo real usando un sistema embebido.

1.2. JUSTIFICACIÓN

La capacidad de cómputo que existe actualmente ha servido para desarrollar gran variedad de algoritmos para el reconocimiento y clasificación de imágenes en tiempo real, los cuales han contribuido enormemente en aplicación dentro de la agricultura de precisión, control de calidad en industrias, entre otras. Dichos algoritmos están soportados sobre sistemas embebidos especializados. Dispositivo como GPUs o FPGAs son la principal fuente de cómputo usadas [3]. Estos sistemas al ser desarrollados con fines específicos plantean una serie de costos considerables que limitan un fácil acceso a ellos. Entre más complejo sea el algoritmo, más tiempo computacional y energía demanda, y si requiere el mejor desempeño posible, debe contar con un buen sistema de hardware [4].

Debido al creciente uso de Deep Learning, diversos frameworks como TensorFlow facilitan la implementación de estos sistemas sobre diversas plataformas incluyendo sistemas embebidos [5]. Existe una variedad de sistemas embebidos en el mercado, sobre la cual el desarrollo de DL ha tenido una muy baja exploración, quizás por una predisposición a que su capacidad de cómputo no sea lo suficientemente buena [6].

Indagar sobre el desempeño que se pueda obtener haciendo uso de este tipo de sistemas, podría crear interés sobre el uso de los mismos para desarrollos futuros, en especial para

aplicaciones de agricultura funcionando en ubicaciones remotas y en tiempo real.

1.3. OBJETIVOS

1.3.1. Objetivo General

Desarrollar e implementar en un sistema embebido, una metodología de clasificación de imágenes en tiempo real, y demostrar su funcionamiento utilizando objetos asociados con aplicaciones agrícolas y/o industriales.

1.3.2. Objetivos Específicos

- Desarrollar un modelo matemático para el problema general de la clasificación de imágenes utilizando técnicas del estado del arte.
- Sintonizar el modelo matemático obtenido para un sistema embebido que permita su operación en tiempo real.
- Diseñar una metodología para la evaluación del desempeño del clasificador embebido, operando en un contexto de aplicación determinado.

2. ESTADO DEL ARTE

La visión por computador tiene como objetivo emular la capacidad de visión de los seres humanos para adquirir información del entorno y poder procesarla en un dispositivo de cómputo para posterior análisis, comprensión, aprendizaje que luego serán utilizadas como solución de problemas del "mundo real" [7].

Con el rápido ascenso de dicha tecnología, se han establecido ciertas metodologías de manejo de información y procesamiento de los datos utilizando algoritmos computacionales que con el tiempo van aumentando en complejidad y robustez pero que incursionan en investigaciones y aplicaciones. El aprendizaje de máquina ha contribuido enormemente como una herramienta fundamental para manejar todos los procesos que los sistemas de visión requieren además de utilizar su principal función de aprender por medio de la experiencia modelos que se pueden ir ajustando de acuerdo a las necesidades y mejorar además su rendimiento óptimo [7].

El aprendizaje de máquina es un subcampo de la inteligencia artificial y se encarga de elaborar algoritmos computacionales para que la máquina aprenda, esté en la capacidad de realizar tareas y que estas puedan ayudar en la resolución de problemas con la experiencia obtenida [8]. Dichos algoritmos se dividen en aprendizaje supervisado y no supervisado; en el aprendizaje supervisado se genera un algoritmo a partir de unos datos de entrenamiento que tienen unas entradas y una salida deseada, todos previamente conocidos [9].

Dentro de las técnicas que comprende el aprendizaje supervisado se encuentra la clasificación, donde la salida deseada es una etiqueta de clase y el objetivo principal es que el algoritmo esté en la capacidad de decidir correctamente a cuál clase pertenece una nueva observación o dato con base en un modelo previamente entrenado [10].

Existen diferentes tipos de clasificadores que se adaptan de acuerdo a la aplicación,

dentro de los más conocidos debido al alto desempeño, buen historial de aplicaciones, haciendo énfasis en clasificación de imágenes y tasa de errores mínimas son la Máquina de Vectores de Soporte (SVM) y las Redes Neuronales [11]. Ambas pertenecen a un conjunto de algoritmos clásicos de Machine Learning y Deep Learning respectivamente. Las redes neuronales han revolucionado el mundo de la inteligencia artificial, porque han permitido el desarrollo de múltiples aplicaciones de manera óptima. En el caso de clasificación de imágenes, las redes neuronales pueden trabajar en modelos complejos y hacer máximo provecho de los datos al tener diferentes formas de extracción automática y aprendizaje de características teniendo como resultado predicciones precisas y con tiempos computacionales mucho menores. Existen diferentes tipos de redes neuronales, donde varía por ejemplo su arquitectura de red como por ejemplo las redes neuronales convolucionales; cada una tiene sus propias funciones y son adaptables a las necesidades del usuario.

La etapa de clasificación tiene un procedimiento general sin importar el algoritmo seleccionado, donde se debe hacer la selección del conjunto de datos en subconjuntos de entrenamiento, validación, prueba; pasar por una extracción de características de los datos usando descriptores de forma o color, después entrenar el modelo teniendo en cuenta ajustes de parámetros con los resultados de la validación y por último hacer la prueba final que determine si se clasifica o no correctamente [10]. Se deben aplicar además métricas para evaluación de desempeño del clasificador que arrojen el mínimo error posible.

De acuerdo a lo anterior, se requiere entonces un dispositivo capaz de procesar el modelo clasificador obtenido además de permitir la operación en tiempo real de reconocimiento y clasificación de nuevas imágenes que recibirá el modelo [12].

Para esto es necesario acudir a la reciente tecnología de sistemas embebidos con procesadores gráficos más conocido como GPU los cuales tienen como objetivo acelerar

procesos computacionales complejos con un consumo de energía bajo y costos asequibles. Un ejemplo claro es la tecnología NVIDIA® Jetson Nano™ que proporciona un kit de desarrollo que se especializa en tareas de inteligencia artificial, particularmente en clasificación de imágenes y detección de objetos [13].

Para sintonizar el modelo y realizar el proceso de clasificación en tiempo real con el sistema embebido es necesario utilizar ciertos frameworks como TensorFlow que ya estén optimizados para estas tareas [14].

3. MARCO TEÓRICO

3.1. VISIÓN POR COMPUTADOR

3.1.1. Aprendizaje Supervisado

En el campo del aprendizaje de máquina, específicamente en el reconocimiento de patrones, se tiene un tipo de metodología conocida como aprendizaje supervisado, donde se fundamentan en conocer previamente los datos de entrenamiento y la salida deseada para desarrollar un modelo que permita realizar una tarea determinada como por ejemplo la clasificación. Dicha tarea, permite asignar un dato de prueba no conocido de modo que el modelo previamente entrenado tenga la capacidad de asignar automáticamente a una clase o etiqueta [15]. Como algoritmos clasificadores ampliamente conocidos se tienen las Máquinas de vectores de soporte (SVM), K-nearest neighbours (KNN) y redes neuronales.

3.1.2. Clasificación

Para comprender mejor el funcionamiento de la clasificación, se tiene por ejemplo un conjunto de datos de círculos y estrellas, el cual le asignaremos un conjunto universal F. Dicho conjunto se divide en dos subconjuntos: el primero de entrenamiento, el cual llamaremos E que se debe conformar por la mayoría de los datos y el segundo, en un conjunto de prueba P con los datos restantes. Adicional a ello, se elige un conjunto de validación V que servirá para ajuste de hiperparámetros.

Después de tener un conjunto de datos separado adecuadamente, se desarrolla una extracción de características numéricas necesaria para el entrenamiento del modelo con los datos del conjunto E. Dicho vector de características puede ser color (en pixeles),

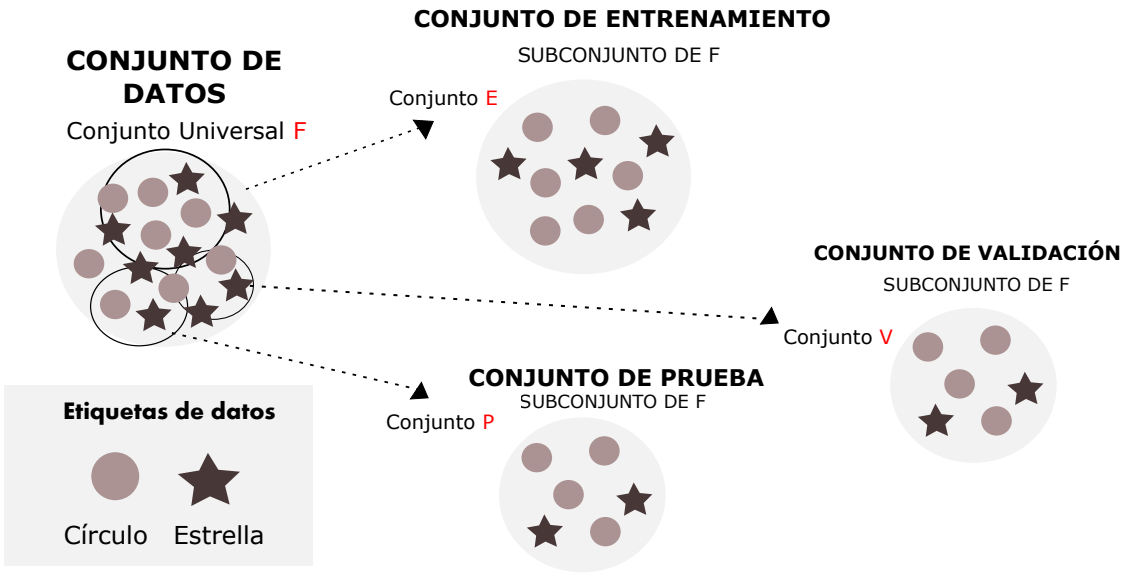


Figura 1. Conjunto de entrenamiento, prueba y validación.

morfología del objeto, pesos, entre otros. El algoritmo de clasificación realiza los cálculos matemáticos necesarios para obtener un modelo predictivo que después de modificar los hiperparámetros, permita probar con los datos de prueba para así obtener un resultado final (etiqueta o clase).

3.2. Extracción de características

La extracción de características, consiste en un proceso de transformación o reducción de los datos (por ejemplo de una imagen), de modo que se obtenga la información más útil y sea más manejable para su uso posterior en detección de objetos o reconocimiento de imágenes. Uno de los descriptores más conocidos es el Histograma de gradientes orientados (HoG), donde se enfoca en calcular la dirección de cada gradiente del píxel de la imagen y sus componente para luego ser colocada en un histograma y analizar los cambios de los datos obtenidos. En el caso de las redes neuronales, se tienen las

convoluciones como extractores de datos, de modo que aplican filtros para detectar bordes o esquinas que son relevantes en la imagen.

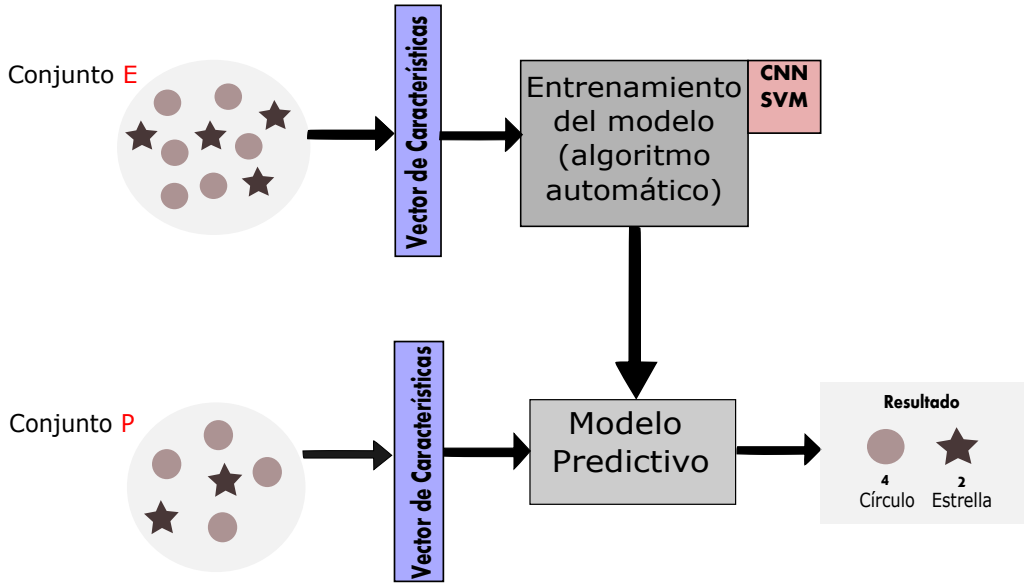


Figura 2. Esquema general de clasificación.

3.3. ALGORITMOS CLÁSICOS DE CLASIFICACIÓN

3.3.1. K- Vecinos Cercanos

El algoritmo KNN *(k-Nearest Neighbour)* o en español *K-Vecinos Cercanos*, es una técnica de clasificación supervisada que se basa en determinar a que clase pertenece un nuevo caso teniendo en cuenta la totalidad de sus K-Vecinos más cercanos [16].

El funcionamiento de este algoritmo depende del cálculo de distancias entre el dato a clasificar y el resto de datos de entrenamiento, para posterior selección de los k elementos con menor distancia y dependiendo de la mayoría que dominen, se le asigna a la clase o etiqueta que correspondan. Generalmente, se utiliza la *Distancia Euclidiana* para determinar la trayectoria más corta posible entre dos puntos, como se muestra en la

ecuación 1, donde $\Delta x1$ es un vector de los datos de entrenamiento y $\Delta x2$ es un vector de las etiquetas de las clases.

$$dx1x2 = \sqrt{\Delta x1^2 + \Delta x2^2} \tag{1}$$

Cabe destacar que la elección de K depende del tamaño del dataset y para ello es necesario elaborar varias pruebas de desempeño del modelo.

3.3.2. Máquinas de Vectores de Soporte

Las máquinas de vectores de soporte o SVM *(Support Vector Machines)* son modelos de clasificación en el campo de aprendizaje supervisado normalmente relacionadas con clasificación binaria o biclase; sin embargo existen variantes del mismo que soportan clasificación multi-clase. El objetivo de SVM, es encontrar un hiperplano que maximiza un margen de separación entre clases [15].

Hiperplano: Para 2 dimensiones, un hiperplano es una línea de separación donde los puntos de soporte están al límite del margen [17].

Dependiendo del hiperplano, se categorizan los datos en un espacio y el modelo debe estar en la capacidad de decidir a que clase pertenece. Para la construcción del hiperplano, se consideran los *Vectores de soporte*, los cuales son puntos que definen el máximo de separación de este. Tienen n elementos y n dimensiones (multidimensionales).

Según la ubicación de los datos, las clases no siempre son linealmente separables, por lo que existe un método llamado kernel que permite transformas las dimensiones para separar los datos con superficies de decisión. Algunos kernel conocidos son: polinomial, sigmoide o rbf.

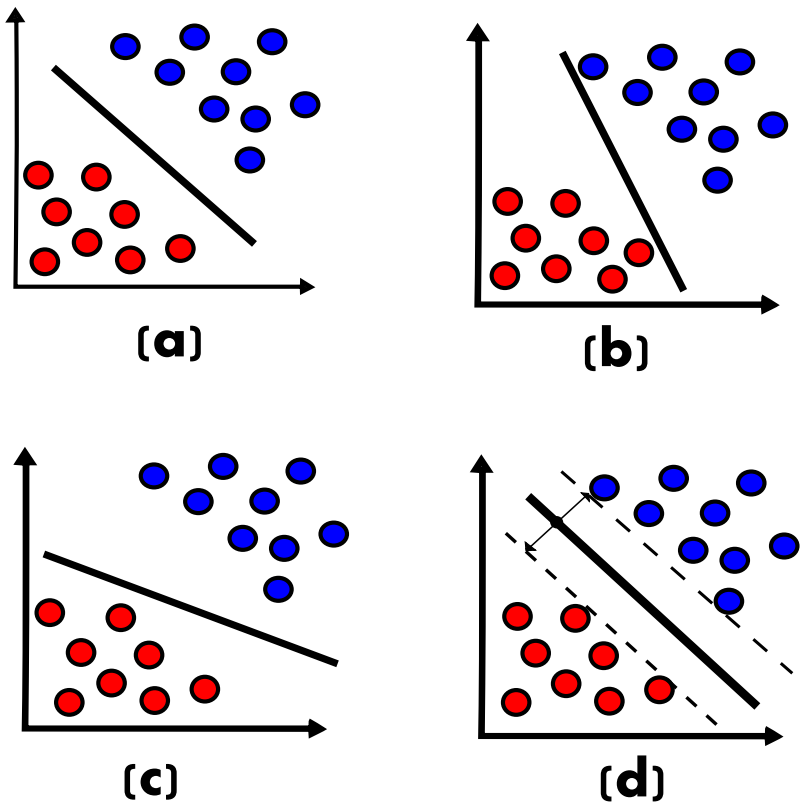


Figura 3. Ejemplo de hiperplanos que separan dos clases en un espacio bidimensional. (a)-(c) Diferentes hiperplanos. (d) Hiperplano de margen máximo [15].

3.4. REDES NEURONALES

Es un método ampliamente utilizado en el campo del aprendizaje profundo, donde la neurona artificial como base fundamental, tiene un comportamiento similar a una neurona biológica.

Una neurona biológica se compone de los siguientes elementos fundamentales: dendritas, cuerpo de la célula donde se encuentra el núcleo, el axón y la sinapsis que permite conectar el axón con las dendritas de otras neuronas. Las dendritas, se encargan de recibir la información para ser procesadas por el cuerpo de la neurona y finalmente ser

transmitidas a otras neuronas por medio del axón [18].

Una neurona artificial es análoga a una neurona biológica. La estructura general es la siguiente:

- **Capa de entrada**: Esta posee n capas de entrada (equivalente a las dendritas) que reciben la información proveniente de otras neuronas. Los datos suelen ser normalizados para obtener mejor precisión numérica en las operaciones que realice la red neuronal [19].

- **Capa oculta:** son capas intermedias análogas al cuerpo de la neurona, donde se realiza la mayoría de procesamiento interno (sumatoria de las entradas).

- **Capa de salida**: Análogo al axón de una neurona biológica, la capa de salida se compone de neuronas y produce un resultado final proveniente del procesamiento en la capa oculta anterior a ella.

Cuando se realiza la sumatoria de las entradas, si esta supera un umbral definido, se aplica una función de activación que posteriormente producirá una salida que a su vez será la nueva entrada de la siguiente neurona [20].

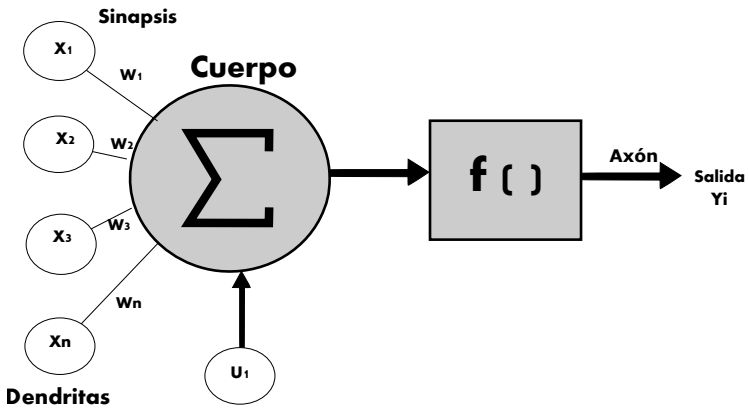


Figura 4. Estructura de una neurona artificial [20].

Un conjunto de neuronas se conoce como red neuronal. Existen redes neuronales con una sola capa donde una sola neurona recibe múltiples entradas y también de múltiples capas, donde varias neuronas reciben varias entradas.

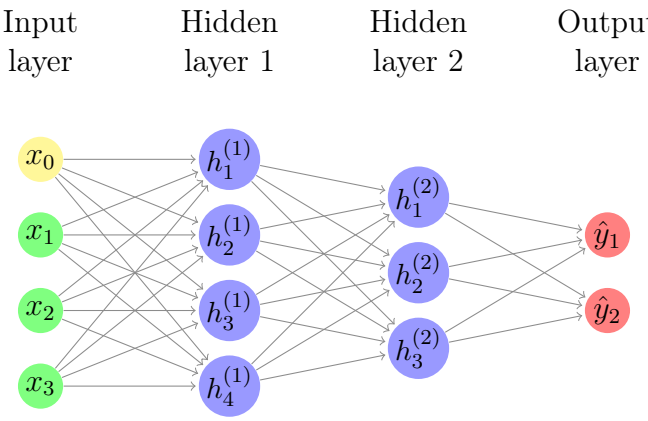


Figura 5. Estructura de una red neuronal multicapa artificial.

Redes Neuronales Convolucionales

Una red neuronal convolucional también conocida como *Convolutional Neural Network (CNN)*, es un tipo de red neuronal multicapa pero que presenta variaciones en las capas, ya que estas realizan una operación conocida como convolución.

Convolución: Una convolución es básicamente un filtro de pesos que se usa para multiplicar un píxel con sus vecinos para obtener un nuevo valor para este. Opera dos matrices, una es la de entrada (matriz normalizada) y la otra una matriz llamada kernel que puede variar de tamaño o adaptarse al tipo de imagen.

Las redes neuronales convolucionales se componen de las siguientes capas:

- **Capa de entrada:** donde se recibe la imagen en forma de matriz de 2 dimensiones para representar los píxeles y donde comúnmente se normaliza a un tamaño que permita un procesamiento.

- **Capas convolucionales:** Son las encargadas de detectar y extraer las características de la imagen en diferentes posiciones. Estas aplican filtros para mejorar la imagen y detectar bordes o esquinas fácilmente, para así obtener un mapa de características que serán entregadas a la siguiente capa que se conoce como capa de agrupamiento (pooling layer).

- **Capa pooling o capa de agrupamiento**: tiene la función de reducir el tamaño de la imagen (resolución) o mapa de características que se obtuvieron en la capa de convolución, con el fin de mantener mas compacta la información.

- **Función de activación:** De acuerdo al número de entradas que recibe la red, define el tipo de salida de la misma, teniendo la función ReLU y Sigmoide de las más notables.

- **Capa densa:** Conecta la capas de la red y clasifica los datos obtenidos anteriormente con un proceso de conversión, que permita obtener un solo vector unidimensional.

- **Capa de salida:** Muestra una predicción final respecto a la imagen recibida y todo el proceso que condujo con las capas anteriores. Dependiendo del ajuste de hiperparámetros y manejo de las convoluciones, puede afectar o no la probabilidad de éxito [21].

3.5. CIFAR-10

El conjunto de datos público CIFAR-10, es ampliamente utilizado para probar diferentes tipos de algoritmos de aprendizaje de maquina en el problema de clasificación general. Consta de 60000 imágenes a color (RGB) de 32x32 con 10 categorías entre objetos y

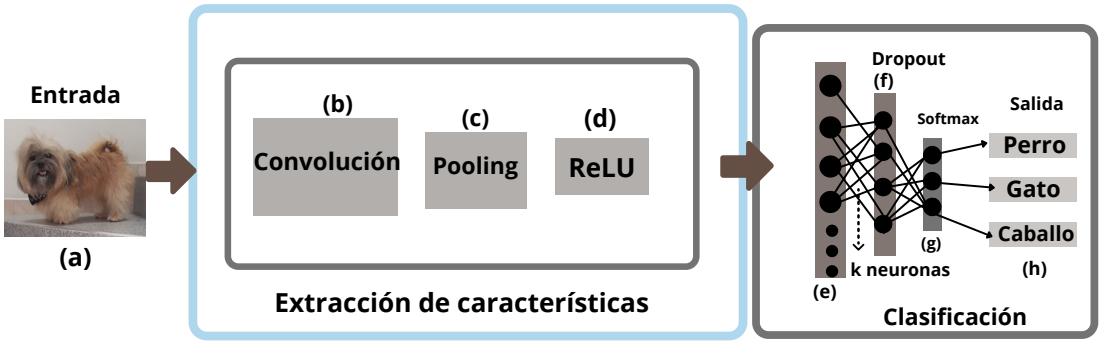


Figura 6. Estructura de una red neuronal convolucional.

animales debidamente etiquetadas, obteniendo 6000 por clase. Del total de imágenes, 50000 son usadas para entrenamiento, y 10000 para prueba.

El conjunto de datos está separado en cinco archivos diferentes como data_batch para los datos de entrenamiento y un archivo test_batch para prueba. Cada archivo, es un objeto *'pickled'* de Python producido con cPickle, este método da la facilidad de representar la información como una cadena de bytes (serialización). La rutina para abrir el archivo se encuentra disponible en la página oficial de CIFAR-10 [22].

Hecho el proceso anterior con cada uno de los archivos se obtiene un diccionario con los siguientes elementos:

Data: Una matriz de 10000x3072 de uint8s. Cada fila de la matriz almacena una imagen en color de 32x32. Adicional a ello posee tres canales de colores donde cada 1024 entradas corresponden al canal rojo, las siguientes 1024 al verde y las restantes al azul.

Labels: Son las etiquetas de cada clase. Es una lista de 1000 números en el rango de 0-9. Adicionalmente, el conjunto de datos posee un archivo llamado *batches.meta* que contiene un diccionario dedicado a otra lista con los nombres de las etiquetas numéricas anteriores.

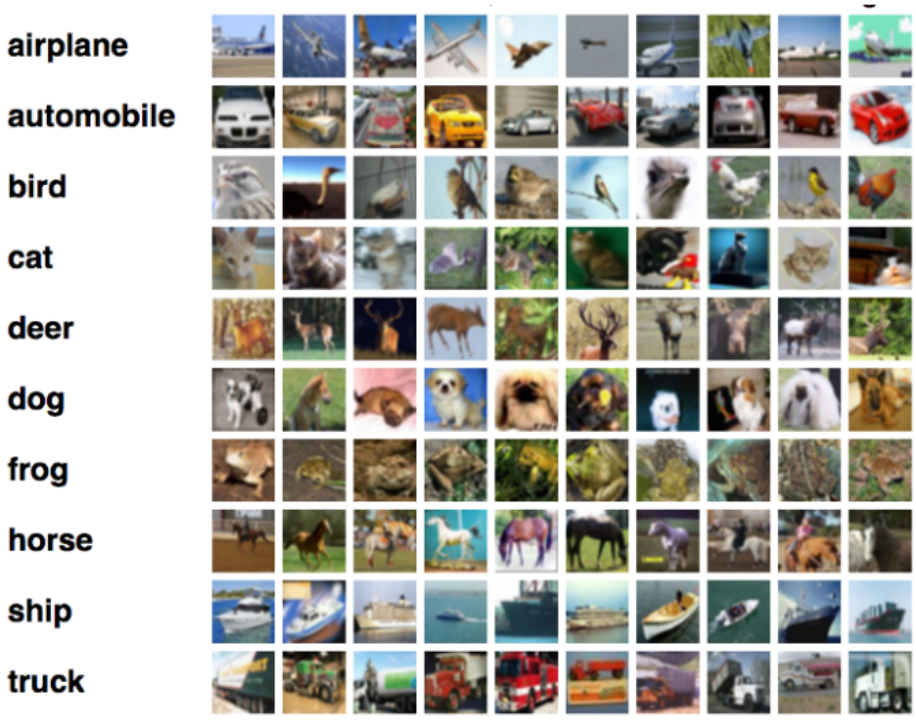


Figura 7. Dataset CIFAR-10 [22].

3.6. RoCoLe DATASET

Es una colección de imágenes de Café Robusta disponible públicamente. Esta es una variedad de café que se cultiva en América del Sur. El conjunto de datos consta de 1560 imágenes. Todas son imágenes sRGB ultra alta resolución (2322×4128).

La base de datos se divide en 6 categorías: (0) es para hojas sanas, las categorías 1 a 4 son para hojas con roya en cuatro niveles de severidad diferentes y la categoría 5 para hojas con ácaros rojos (tipo diferente de infección). En la siguiente figura se puede apreciar algunas muestras de hojas provenientes de la base de datos donde se muestra tanto sanas como enfermas [23].

Figura 8. RoCoLe Dataset [23].

4. SINTONIZACIÓN

4.1. SISTEMA EMBEBIDO

Se diferencian de equipos de cómputo convencionales por la capacidad de procesamiento en tiempo real en aplicaciones específicas como tareas automatizadas, visión por computador o robótica, reemplazando así sistemas robustos y costosos por unos más asequibles, flexibles y fáciles en su manejo y programación.

4.2. GPU

Utiliza una unidad de procesamiento de gráficos para acelerar el funcionamiento de aplicaciones de inteligencia artificial específicamente las que involucren aprendizaje profundo. Poseen más núcleos que procesan en paralelo logrando mayor eficiencia y velocidad en la ejecución de las aplicaciones.

4.3. JETSON NANO

El sistema embedido NVIDIA® Jetson Nano™ pertenece a la última generación de sistemas embebidos con capacidad de procesamiento paralelo en GPU. Proporciona un kit de desarrollo que se especializa en tareas de inteligencia artificial, particularmente en clasificación de imágenes y detección de objetos. Además es un dispositivo pequeño que posee varios puertos de entrada para conectar pantalla, teclado, mouse y cámara necesarios para el desarrollo del proyecto.

4.3.1. Instalación en Sistema Embebido

NVIDIA® Jetson Nano™ cuenta con un sistema operativo Linux

Para iniciar con la instalación es necesario una tarjeta microSD como almacenamiento principal y en el cual se instalará todo lo necesario. Se escogió una con referencia UHS-1 de 32 GB, la cual recomienda el fabricante.

Siguiendo las instrucciones dadas por el fabricante en su página oficial [13], se descarga la imagen del sistema operativo jetpack 4.2 basado en Ubuntu sobre la microSD que tuvo que ser formateada con anterioridad. Se descarga e instala balenaEtcher, la cual es una herramienta que graba la imagen del sistema operativo en una SD conectada a un ordenador personal. A continuación, se conecta la microSD en la Jetson Nano con todos los componentes necesarios adicionales: se utilizó una fuente de alimentación de barril de 5V 2.5A con puente en el conector J48, así como también un teclado y mouse (USB) , un módulo de conexión a WiFi inalámbrico (USB) y una pantalla con puerto HDMI.

Se enciende la Jetson Nano y se realizan las configuraciones iniciales necesarias como idioma, usuario, memoria a usar y conexión a internet. Lo siguiente a realizar será la instalación de todas las dependencias necesarias para el desarrollo del proyecto como Python, TensorFlow, Keras, OpenCV, NumPy, entre otros.

4.3.2. Adaptación del modelo de clasificación en el sistema embebido

Después de tener instaladas las librerías correspondientes, se procede con la adaptación del modelo de clasificación más eficiente, el cual corresponde a las redes neuronales convolucionales.

Dicho modelo fue entrenado en Google Colaboratory y guardado en cuanto a los valores de pesos (obtenidos en el entrenamiento) y el optimizador dentro de un archivo con el módulo model.save(ruta.h5") que provee TensorFlow en su documentación oficial.

Hecho esto, se procedió a crear un programa python en la Jetson Nano el cual contenga

la arquitectura de la red neuronal convolucional igual a la ya entrenada, una interfaz que me permita elegir el medio de obtención de la imagen, ya sea capturada por una cámara o leída desde la Jetson Nano, además del archivo donde se guardaron los pesos del modelo entrenado. Para cargar el modelo (nombrado como Saved_model.h5) se necesita de la instrucción model.load_weights(/Saved_model.h5").

Hecho esto se procede a realizar la predicción de la imagen por medio de la instrucción predict_proba pasando la imagen que debe ser convertida a un arreglo de numpy y con una resolución menor a las imágenes originales para obtener mejor desempeño y evitar el agotamiento de la máquina.

Como paso final se obtiene la probabilidad y el porcentaje de acierto para la hoja sana o enferma.

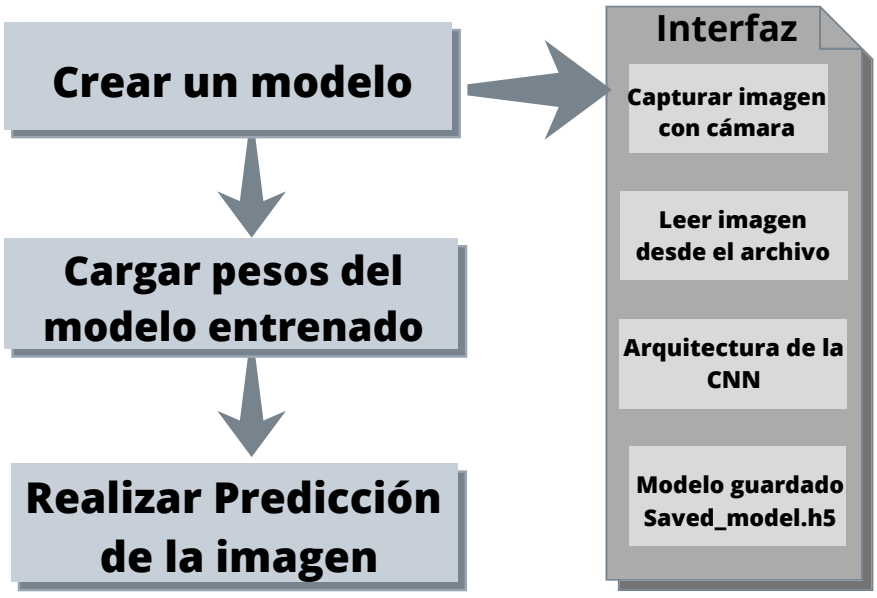


Figura 9. Proceso de adaptación del modelo en el sistema embebido.

4.4. FRAMEWORK (TENSORFLOW)

TensorFlow es una plataforma de código abierto que contiene modelos de aprendizaje automático, y ofrecen distintas herramientas para desarrollar aplicaciones de aprendizaje profundo e implementarlos en sistemas embebidos, dispositivos móviles o GPUs con el fin de acelerar procesos de ejecución. Además, posee una API llamada Keras que facilita el manejo de datos y de modelos complejos como las redes neuronales, simplificando el proceso que conlleva, siendo así un sistema versátil y flexible para el usuario.

5. EVALUACIÓN

5.1. MÉTRICAS DE DESEMPEÑO

5.1.1. Validación

El conjunto de validación es un subconjunto de datos separado del conjunto de entrenamiento que se utiliza para ajustar hiperparámetros. Estos son valores que el usuario ajusta durante las ejecuciones del entrenamiento de modo que se pueda obtener el mejor desempeño posible con los datos de validación antes de utilizar los datos de prueba.

5.1.2. Validación cruzada

El conjunto de datos es dividido en partes iguales. En cada partición el algoritmo se entrena usando todas menos una de las particiones y se prueba con la restante. Este procedimiento iterativo se realiza sobre todas las particiones obteniendo finalmente alguna métrica para evaluar el modelo [15]. La ventaja de usar validación cruzada es que ayuda a proporcionar una estimación más precisa del desempeño del modelo que solo usando una única partición de los datos.

5.1.3. Precisión

Se define como el porcentaje o cantidad de datos acertados correctamente sobre el total de muestras a predecir. Se llaman verdaderos positivos (VP) o verdaderos negativos (VN) aquellos datos que se clasifican o aciertan correctamente y falsos positivos (FP) o negativos (FN) a los que no. De acuerdo a la definición anterior, a precisión se define entonces de la siguiente manera:

$$accuracy = \frac{VP + VN}{VP + VN + FP + FN} * 100\,\% \tag{2}$$

5.1.4. Matriz de Confusión

La matriz de confusión da un estimado de la cantidad de verdaderos o falsos (sean positivos o negativos) de modo que se pueda verificar la eficacia del modelo. Lo ideal es que se tengan solo verdaderos positivos y negativos para obtener un excelente desempeño. Adicionalmente, existen herramientas como sklearn donde se puede implementar la matriz de confusión de una manera sencilla en modelos de clasificación simplemente pasando como argumento los datos y etiquetas de predicción.

		PREDICCIÓN	
		POSITIVO	NEGATIVO
OBSERVACIÓN	POSITIVO	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	NEGATIVO	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 10. Matriz de Confusión.

5.1.5. Pérdidas

La pérdida de entrenamiento *Loss training* y pérdida de validación *Validation loss*, corresponden a los errores en el conjunto de datos de entrenamiento y en la ejecución del conjunto de datos de validación con la red entrenada respectivamente. En las redes

neuronales, ayudan a indicar si el modelo presenta sobreajuste, observando el comportamiento de ambos errores cuando aumentan las épocas de entrenamiento. Esto sucede, cuando loss training sigue disminuyendo y validation loss comienza a aumentar.

6. EXPERIMENTOS Y RESULTADOS

Los experimentos realizados en este trabajo constan en desarrollar modelos de clasificación con 3 diferentes algoritmos (SVM, KNN y CNN) utilizando como base de datos CIFAR-10 para así verificar el mejor desempeño y así aplicar la metodología desarrollada para el entrenamiento y modelamiento de los datos de la aplicación seleccionada (agricultura) usando la base de datos de café RoCoLe. Posteriormente, se determinan los resultados de desempeño de esta, tanto en entrenamiento y validación como durante la ejecución en tiempo real con redes neuronales en el sistema embebido (Jetson Nano).

6.1. CIFAR-10

Para esta base de datos multiclase, se realizaron dos modelos clásicos de clasificación con el objetivo de comprender su funcionamiento con una base de datos extensa ya establecida, además comparar su desempeño y tiempo de ejecución.

6.1.1. KNN

Se diseñó primero una etapa de extracción de características, teniendo como primera medida la conversión de las imágenes RGB a escala de grises, para posteriormente usar el descriptor HOG. Hecho esto, se desarrolló el modelo usando el módulo de clasificación n_neighbors de la librería Sk-learn. Para hallar los (*k vecinos cercanos más optimos*), se implementó validación cruzada dividiendo los datos de entrenamiento en 5 folds. Los resultados se muestran en la Tabla 1.

Tabla 1. Resultados obtenidos de validación cruzada (KNN) para CIFAR-10.

k vecino cercano	Precisión
2	0.4391
5	0.4949
8	0.5029
10	0.5073
12	0.506
15	0.5045

6.1.2. SVM

Se utilizó un clasificador lineal con entrenamiento SGD (Stochastic Gradient Descent) de la librería SK-Learn como una solución más óptima para el tiempo de ejecución. Se establecieron como argumentos importantes una función de pérdida logarítmica, una constante alpha de 0.04 y como hiperparámetro variable, el máximo de iteraciones que modifica el comportamiento de los datos de entrenamiento, donde se obtuvo el mejor resultado con max_iter = 50 .

Tabla 2. Resultados obtenidos para SVM con la base de datos CIFAR-10.

max_iter	Precisión
20	0.4256
50	0.431
100	0.4246
150	0.4281

6.1.3. CNN

Se implementó una red neuronal convolucional sencilla para probar el rendimiento y compararla con los anteriores algoritmos. La topología de la red consta de tres capas convolucionales (Conv2D), de tamaño 3x3 y función de activación ReLU, teniendo dos de ellas 32 mapas de características y la restante con 64. Adicionalmente, se tiene un par de capas MaxPooling de 3x3, una capa Flatten y dos capas totalmente conectadas

usando nuevamente la función de activación ReLU. Para finalizar, se compila el modelo con métricas de desempeño como *'accuracy*, pérdidas *'SparseCategoricalCrossentropy'* y se entrena para 10 épocas.

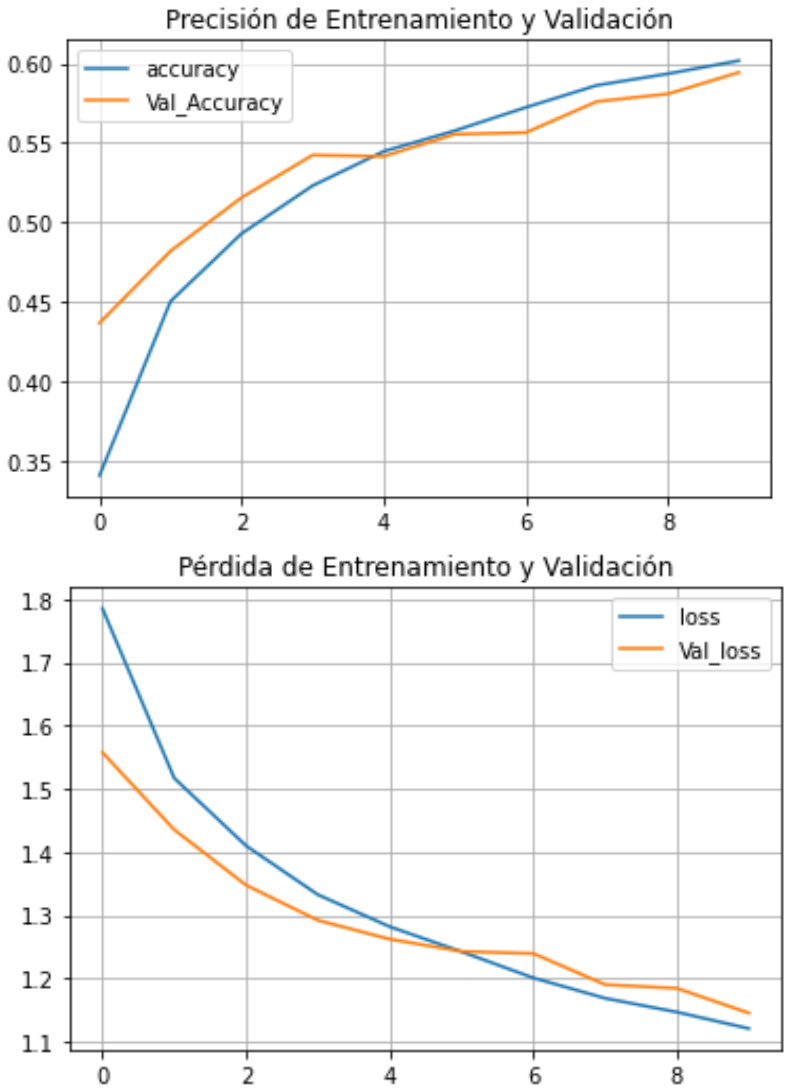


Figura 11. Precisión de entrenamiento y prueba del modelo para CIFAR-10.

Como resultado, se obtiene para entrenamiento una precisión del 60 % con pérdida de 1.18 y para validación una precisión del 59.43 % con pérdida de 1.1451.

Por su parte la prueba o test muestra una precisión del 59.43 % con una pérdida de 1.1451, lo cual comparado a los demás algoritmos demuestra un buen desempeño pese a estar conformada la red por una arquitectura sencilla.

Los resultados de los tres modelos implementados utilizando la base de datos CIFAR-10 se muestran a continuación:

Tabla 3. Resultados de clasificación en CIFAR-10.

	Tiempo ejecución Train [s]	Tiempo ejecución Test [s]	Precisión
KNN	3.81	430.175	0.5073
SVM	9.896	0.01466	0.431
CNN	58.32	0.955	**0.5943**

6.2. ROCOLE- DATASET

Como se había mencionado, esta base de datos es mas pequeña que CIFAR-10, además se predecir solo 2 clases, por lo que se hizo variaciones para el algoritmo de SVM y se agregó la red neuronal convolucional para demostrar su desempeño en la fase de entrenamiento y prueba e implementada sobre el sistema embebido en tiempo real. Adicionalmente se realizó un reajuste de escala para las imágenes de 2048 x 1152 píxeles (alto y ancho respectivamente) a una de 200 x 120 porque permiten mejor desempeño y mejoran el tiempo de ejecución.

6.2.1. KNN

Se ajustó la base de datos para el modelo previamente desarrollado, realizando una separación de datos de 1403 x 72000 imágenes de entrenamiento y 157 x 72000 para datos de prueba. Se realiza la extracción de características usando el descriptor HOG, teniendo en cuenta una conversión previa de las imágenes a escala de grises. Se entrena el modelo usando el módulo de clasificación n_neighbors de la librería Sk-learn. La

validación cruzada para la elección del mejor hiperparámetro dio como resultado k=2 como se muestra en la tabla, para así obtener una precisión de 0.834 en el modelo.

Tabla 4. Resultados obtenidos de validación cruzada para RoCoLe.

k vecino cercano	Precisión
2	0.8343
3	0.7834
7	0.6751
9	0.63964
13	0.5923
16	0.63057
20	0.6178

6.2.2. SVM

Para este modelo se consideró la librería Sk-learn con el clasificador de máquinas de vectores de soporte SVC, entrenando con Gradient Descent (GD) sin la necesidad de implementar SGD ya que esta base de datos consta de menos imágenes, por lo que no resultaron problemas de memoria. Este tiene como hiperparámetro principal el kernel, del cual se obtuvo un mejor desempeño con *'rbf'* que suele ser el predeterminado por la función.

6.2.3. CNN

Para implementar la red Neuronal convolucional se utilizó la librería TensorFlow y Keras para el preprocesamiento de datos y fácil manipulación de estos. Las imágenes siguen con el reajuste de escala de 200 x 120 y divididas en 1403 x 72000 imágenes de entrenamiento y 157 x 72000 para datos de prueba.

La red Neuronal consta de una capa convolucional de entrada que toma tensores de forma (alto, ancho, canales de color) de las imágenes, siendo para este caso (200,120,3)

donde los canales de color se refiere a RGB; dos capas Conv2D que contienen la función de activación ReLU para filtrar los valores positivos de los negativos y dos capas MaxPooling2D que disminuyen a la mitad la resolución de las imágenes. Se tiene además una capa de aplanamiento y como es un problema de clasificación binaria, se usa una función de activación sigmoide, de modo que la salida de la red será un escalar entre 0 y 1. Finalizada la arquitectura, se compila el modelo utilizando como Optimizador *'Adam'*, función de pérdida *'binary_ crossentropy'* y *'accuracy'* como métrica de desempeño. Para el entrenamiento del modelo se utiliza como hiperparámetro *'epochs=15'* y un *'batch-size'*=1.

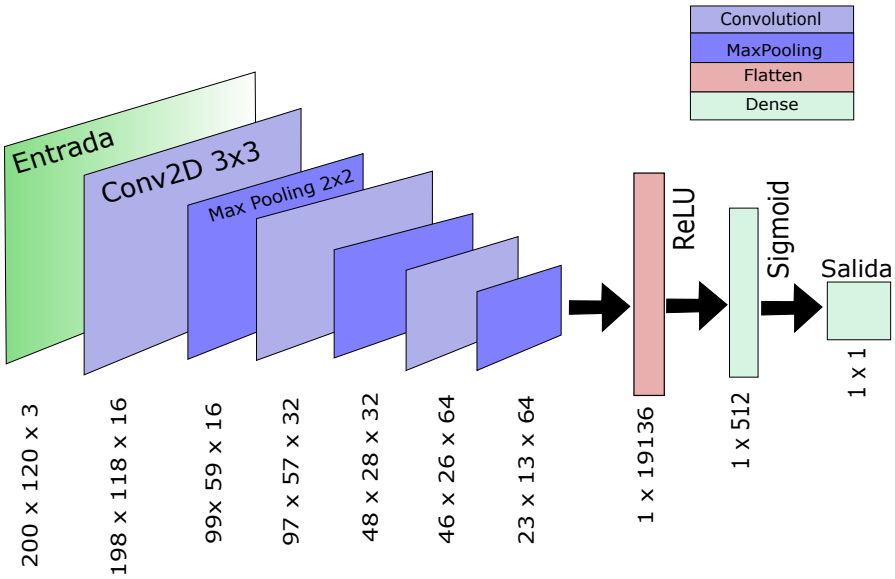


Figura 12. Arquitectura de la CNN implementada sobre RoCoLe.

Por último, se realizó la evaluación del modelo donde se observa su desempeño para validación y entrenamiento, donde se obtuvo una precisión del 98.21 % para entrenamiento con pérdida de 0.0154 y para validación con una precisión del 99.36 % con una pérdida de 0.0342. Adicionalmente, se obtuvo una precisión del 99.36 % con una pérdida de 0.0342 con un tiempo de ejecución de 0.933 s para Test, demostrando así un excelente desempeño del modelo.

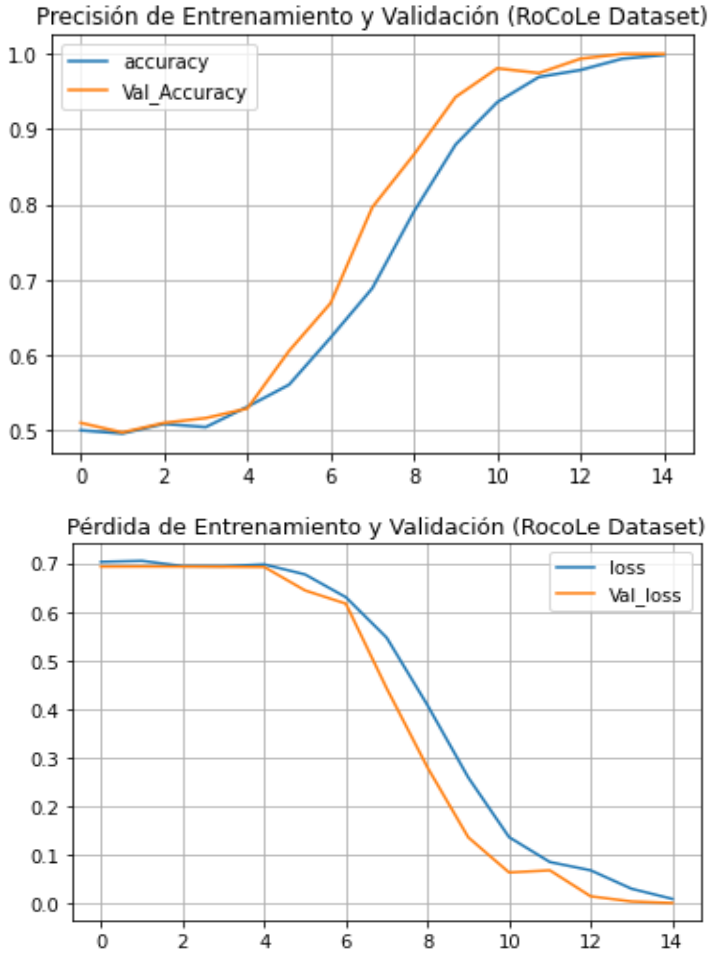


Figura 13. Precisión para validación y entrenamiento de la CNN para RoCoLe.

A continuación, se muestra la comparación entre los 3 modelos implementados para la base de datos RoCoLe:

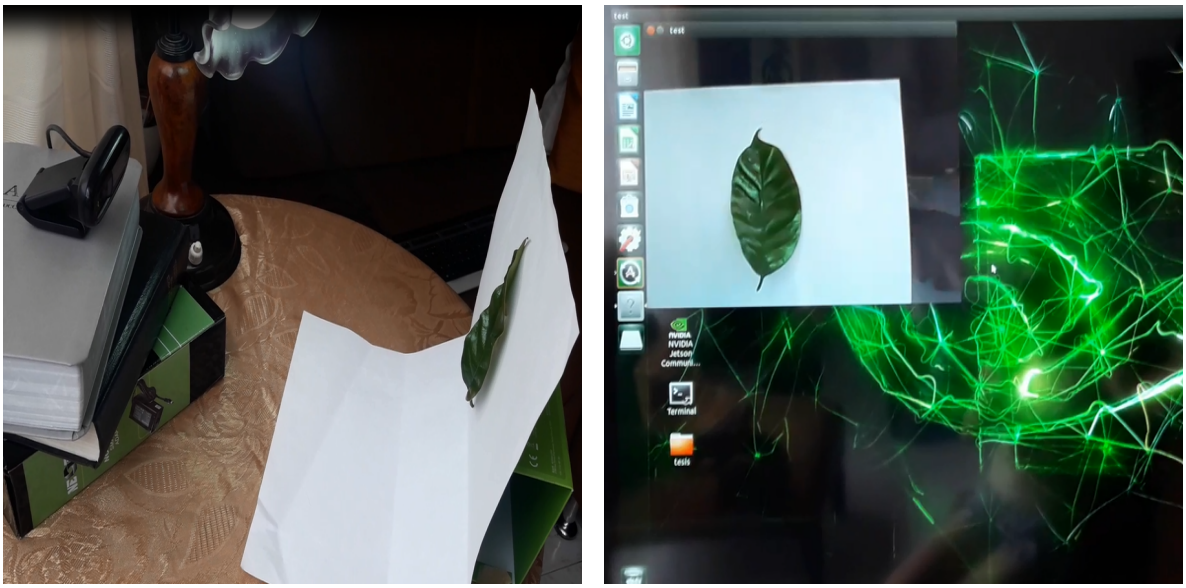
Tabla 5. Resultados de clasificación en RoCoLe.

	Tiempo ejecución Train [s]	Tiempo ejecución Test [s]	Precisión
KNN	1.463	6.088	0.834
SVM	38.796	4.38105	0.96178
CNN	124.515	0.933	**0.9936**

6.3. Evaluación del desempeño del clasificador

En la implementación del modelo en el sistema embebido, se realizaron distintas pruebas de desempeño utilizando imágenes nunca antes vistas por el modelo, tomadas por una cámara en tiempo real o cargadas de una carpeta local. El código desarrollado tiene una interfaz que permite decidir cual de estos métodos aplicar.

Se usó una cámara web comercial marca Logitech, de referencia C170, con 5 megapíxeles y fotografías con resolución de 640 x 480 píxeles. Las imágenes capturadas fueron reajustadas de escala a 200x120 para no agotar la memoria del nVidia Jetson Nano. Adicionalmente se acondicionó luz artificial para mejorar la calidad de la imagen y se mostraron las hojas captadas por la cámara en una pantalla Figura 14 (b).



(a) Implementación de cámara. (b) Hoja vista desde el sistema embebido.

Figura 14. Configuración de hardware para realizar las pruebas de clasificación sobre el nVidia Jetson Nano.

Para controlar el flujo del programa, se utilizó un cliente SSH que conecta el computador personal con la Jetson Nano de forma remota como se muestra a continuación.

Hechas las configuraciones necesarias, se realizaron capturas con la cámara de hojas de

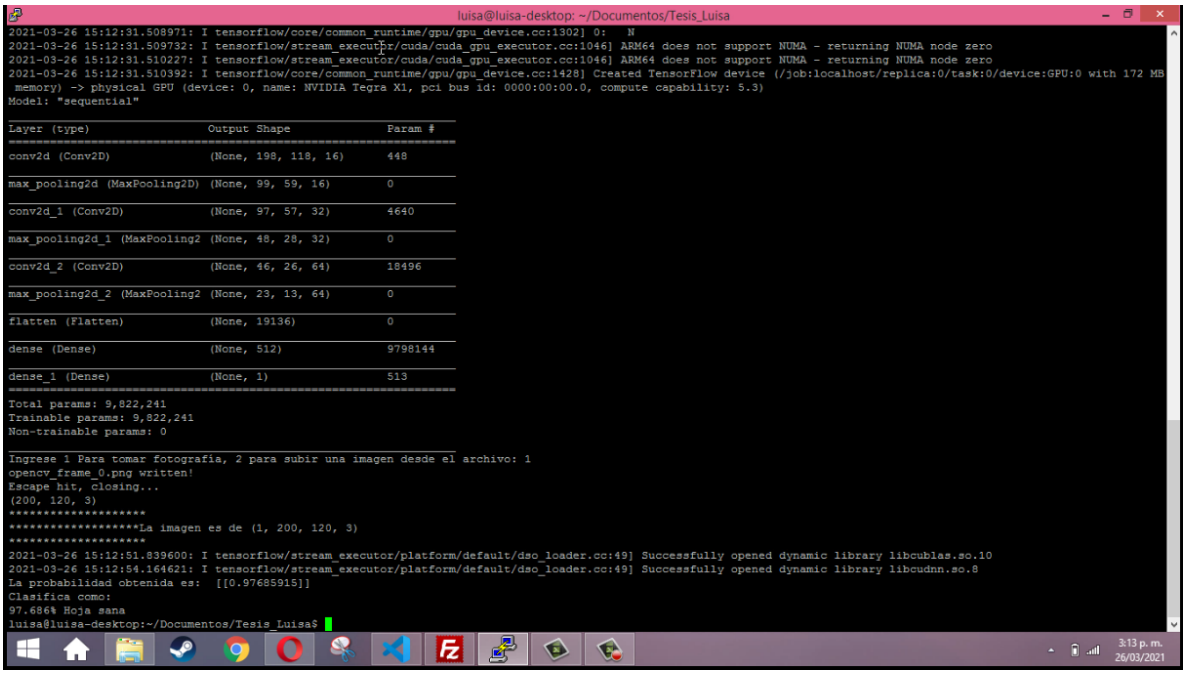


Figura 15. Ejecución del modelo en el sistema embebido.

café sanas y enfermas que se muestran en la figura 16, para así medir el intervalo de confianza de cada predicción.

Tabla 6. Desempeño de clasificación sobre el sistema embebido.

Tipo de hoja	Intervalo de confianza [%]
Hoja enferma (a)	98.876
Hoja sana (b)	97.686
Hoja enferma (c)	74.024
Hoja enferma (d)	79.319

(a) Hoja enferma (archivo local)

(b) Hoja sana captada por la cámara

(c) Hoja enferma captada por la cámara

(d) Hoja enferma captada por la cámara

Figura 16. Hojas de prueba implementadas sobre el sistema embebido.

7. CONCLUSIONES Y RECOMENDACIONES

7.1. CONCLUSIONES

CNN obtuvo un desempeño significativamente mayor que los modelos clásicos kNN y SVM, tanto para la base de datos CIFAR como para RoCoLe. Además, la CNN desarrollada posee un costo computacional controlado, haciendo de ella una elección viable para el sistema embebido.

Hay múltiples modelos de redes neuronales que pueden variar el desempeño final, dependiendo de la topología de la red, el número de épocas en el entrenamiento y la cantidad de imágenes que contenga la base de datos.

Se logró implementar satisfactoriamente el sistema embebido utilizando la tecnología NVidia Jetson Nano. Al utilizar CNN fue necesario instalar un framework de redes neuronales convolucionales en la Jetson, para este caso, TensorFlow, y se desarrolló un app en Python para la evaluación de la red CNN diseñada. La metodología implementada podría extenderse a pruebas futuras usando otro tipo de Frameworks como Pythorch o Caffe para realizar comparación de desempeño.

Se comprobó el correcto funcionamiento del sistema embebido de forma local. Se utilizaron tanto imágenes almacenadas como capturas en vivo desde la cámara del sistema embebido, arrojando resultados satisfactorios en ambos casos.

Aplicaciones industriales y de agricultura de precisión podrán beneficiarse de los nuevos desarrollos en Redes Neuronales Convolucionales al poder correr aplicativos de aprendizaje profundo en sistemas embebidos, sin requerimientos especiales de hardware, y en ubicaciones remotas donde el acceso a la red eléctrica y de datos sea aún escasa.

7.2. RECOMENDACIONES

Es importante que al trabajar con un dataset de imágenes de gran tamaño sean dimensionadas a un tamaño menor, debido a que tanto el equipo donde se entrenen los modelos como el sistema embebido, ejecutaran en menor tiempo y sin presentar problemas de agotamiento en memoria.

Al configurar el NVidia Jetson Nano, se recomienda utilizar una microSD de más de 32 GB para instalar sistema operativo, librerías y dependencias necesarias. Adicionalmente, se recomienda el uso de una cámara comercial de gama media o baja para no presentar problemas con sus controladores.

Se recomienda el uso de un software que permita el acceso remoto desde un ordenador personal al sistema embebido, para mayor comodidad al momento de realizar pruebas o programar.

BIBLIOGRAFÍA

[1] RIZVI, Syed Tahir Hussain; CABODI, Gianpiero y FRANCINI, Gianluca. Optimized Deep Neural Networks for Real-Time Object Classification on Embedded GPUs. En: APPLIED SCIENCES-BASEL, tomo 7, Nº 8, 2017. 1.1

[2] KAMILARIS, Andreas y PRENAFETA-BOLDÚ, Francesc X. Deep learning in agriculture: A survey. En: Computers and Electronics in Agriculture, tomo 147, 2018, págs. 70–90. ISSN 0168-1699. 1.1

[3] VALENCIA SUAREZ, Fabio. GPGPU, un cambio de paradigma para el diseño de programas de alto rendimiento en la Universidad Tecnológica de Pereira y sus áreas de influencia. Tesis de Maestría, Universidad Tecnológica de Pereira. Facultad de Ingenierías, 2013. 1.2

[4] MAZZIA, V., *et al.* Real-Time Apple Detection System Using Embedded Systems With Hardware Accelerators: An Edge AI Application. En: IEEE Access, tomo 8, 2020, págs. 9102–9114. ISSN 2169-3536. 1.2

[5] PASZKE, Adam, *et al.* Automatic differentiation in pytorch. En: NIPS 2017 Workshop Autodiff Submissions, 2017. 1.2

[6] LOPES, Noel y RIBEIRO, Bernardete. GPUMLib: An efficient open-source GPU machine learning library. En: International Journal of Computer Information Systems and Industrial Management Applications, tomo 3, 2011, págs. 355–362. 1.2

[7] SEBE, Nicu, *et al.* Machine Learning in Computer Vision, tomo 29. 1ª edición. Springer Netherlands, 2005. ISBN 978-1-4020. 2

[8] OLIVA, Diego y CUEVAS, Erik. An Introduction to Machine Learning. Springer International Publishin, Cham, 2017. ISBN 978-3-319-48550-8, 1-11 págs. 2

[9] BEKKERMAN, Ron; BILENKO, Mikhail y LANGFORD, John. Scaling up Machine Learning. Cambridge University Press, 2012. 2

[10] SATUÉ, MANUEL GARRIDO. Reconocimiento de señales de tráfico para un sistema de ayuda a la conducción. En: Universidad de sevilla escuela técnica superior de ingeniería, 2013, págs. 13–14. 2

[11] QI, X.; WANG, T. y LIU, J. Comparison of Support Vector Machine and Softmax Classifiers in Computer Vision. En: 2017 Second International Conference on Mechanical, Control and Computer Engineering (ICMCCE), 2017, págs. 151–155. 2

[12] HURTADO, A. F., *et al.* Proposal of a Computer Vision System to Detect and Track Vehicles in Real Time Using an Embedded Platform Enabled with a Graphical Processing Unit. En: 2015 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE), 2015, págs. 76–80. 2

[13] DEVELOPER, NVIDIA. Hardware for every situation, 2019. URL https://developer.nvidia.com/embedded/develop/hardware. 2, 4.3.1

[14] VALDERRAMA MOLANO, Jhony Andres, *et al.* Clasificación de objetos usando aprendizaje profundo implementado en un sistema embebido. Tesis de Maestría, Universidad Autónoma de Occidente, 2017. 2

[15] ALEGRE, E; PAJARES, G y DE LA ESCALERA, A. Conceptos y métodos en visión por computador. En: España: Grupo de Visión del Comité Español de Automática (CEA), 2016, págs. 196–198. 3.1.1, 3.3.2, 3, 5.1.2

[16] MOUJAHID, Abdelmalik; INZA, Inaki y LARRANAGA, Pedro. Tema 5. Clasificadores K-NN. En: Departamento de Ciencias de la Computación e inteligencia artificial, Universidad del País Vasco-Euskal Herriko Unibertsitatea, 2019. 3.3.1

[17] JIMENEZ, Mónica. Acercamiento a las máquinas de soporte vectorial y sus Aplicaciones en proyectos de grado del programa de Ingeniería de Sistemas y Computacion de la Universidad Tecnologica de Pereira. Tesis de Maestría, Universidad Tecnológica de Pereira, 2012. 3.3.2

[18] ACEVEDO, Eder; SERNA, Alexei y SERNA, Edgar. Principios y características de las redes neuronales artificiales. En: DESARROLLO E INNOVACIÓN EN INGENIERÍA, 2017, pág. 176. 3.4

[19] DA SILVA, Ivan Nunes, *et al.* Artificial neural network architectures and training processes. En: Artificial neural networks. Springer, 2017, págs. 21–28. 3.4

[20] CICERO, Ignacio Ezequiel. Utilización de Redes Neuronales convolucionales para la detección de tipos de imágenes. Tesis de Maestría, Instituto Tecnológico de Buenos Aires- ITBA, 2018. 3.4, 4

[21] LÓPEZ-SACA, Fidel, *et al.* Clasificación de imágenes usando redes neuronales convolucionales. Tesis de Maestría, Universidad Autónoma Metropolitana (México). Unidad Azcapotzalco . . . , 2019. 3.4

[22] KRIZHEVSKY, Alex; HINTON, Geoffrey, *et al.* Learning multiple layers of features from tiny images. En: Canadian Institute For Advanced Research, 2009. 3.5, 7

[23] PARRAGA-ALAVA, Jorge, *et al.* RoCoLe: A robusta coffee leaf images dataset for evaluation of machine learning based methods in plant diseases recognition. En: Data in Brief, tomo 25, 2019, pág. 104414. ISSN 2352-3409. 3.6, 8